

Simulation Based Assignment

of

Operating System

**SUBMITTED BY: SUBMITTED TO:**

**TANYA SINGH MS. RITIKA SHARMA**

**ROLL NO. : B35**

**REG NO. : 11614718**

**SECTION: K1640**

**EMAIL ID:** [**tanyasingh2601@gmail.com**](mailto:tanyasingh2601@gmail.com)

**GITHUB LINK:** <https://github.com/tanyasingh-K1640/os_project>

**QUESTION:**

**Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time he wishes to dedicate fixed amount of time to every request so that everyone gets a fair share of his time. He willing into the system from 10am to 12am only. He wants to have separate requests queues for students and faculty. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time.**

CODE:

#include<stdio.h>

#include<stdlib.h>

int handleQueries();

int main()

{

while(1){

int detail;

printf("\nEnter your Occupation : \n1.Faculty \t2.Student \t3.Exit\n");

scanf("%d",&detail);

switch(detail){

case 1:

printf("\n\t\tFaculty Queries : ");

handleQueries();

break;

case 2:

printf("\n\t\tStudent Queries");

handleQueries();

break;

case 3:

printf("\nProgram terminated ");

exit(0);

break;

default:

printf("\nWrong key pressed");

break;

}

}

}

int handleQueries(){

int i,j,n,time,remain,flag=0,timeQuant,detail;

int waitTime=0,turnaroundTime=0;

int at[10],bt[10],rt[10];

printf("\nEnter Total Process:\t ");

scanf("%d",&n);

remain=n;

for(i=0;i<n;i++)

{

printf("\nRequest Number %d :",i+1);

printf("\nEnter Arrival Time and Burst Time of Request");

scanf("%d",&at[i]);

scanf("%d",&bt[i]);

rt[i]=bt[i];

}

printf("\nHow much time each request take ? :\t");

scanf("%d",&timeQuant);

printf("\n\nRequest\t|Turnaround Time|Waiting Time\n\n");

for(time=0,i=0;remain!=0;)

{

if(rt[i]<=timeQuant && rt[i]>0)

{

time+=rt[i];

rt[i]=0;

flag=1;

}

else if(rt[i]>0)

{

rt[i]-=timeQuant;

time+=timeQuant;

}

if(rt[i]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",i+1,time-at[i],time-at[i]-bt[i]);

waitTime+=time-at[i]-bt[i];

turnaroundTime+=time-at[i];

flag=0;

}

if(i==n-1)

i=0;

else if(at[i+1]<=time)

i++;

else

i=0;

}

printf("\nAverage Waiting Time= %f\n",waitTime\*1.0/n);

printf("Avg Turnaround Time = %f",turnaroundTime\*1.0/n);

return 0;

}

Question 1: Explain the problem in terms of operating system concept?

**Answer: Round-robin (RR) is one of the algorithms employed by**[**process**](https://en.wikipedia.org/wiki/Process_scheduler)**and**[**network schedulers**](https://en.wikipedia.org/wiki/Network_scheduler)**in**[**computing**](https://en.wikipedia.org/wiki/Computing)**.**[**[1]**](https://en.wikipedia.org/wiki/Round-robin_scheduling#cite_note-ostep-1-1)[**[2]**](https://en.wikipedia.org/wiki/Round-robin_scheduling#cite_note-Zander-2)**As the term is generally used,**[**time slices**](https://en.wikipedia.org/wiki/Preemption_(computing)#Time_slice)**(also known as time quanta)**[**[3]**](https://en.wikipedia.org/wiki/Round-robin_scheduling#cite_note-3)**are assigned to each process in equal portions and in circular order, handling all processes without**[**priority**](https://en.wiktionary.org/wiki/priority)**(also known as**[**cyclic executive**](https://en.wikipedia.org/wiki/Cyclic_executive)**). Round-robin scheduling is simple, easy to implement, and**[**starvation**](https://en.wikipedia.org/wiki/Resource_starvation)**-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. It is an**[**operating system**](https://en.wikipedia.org/wiki/Operating_system)**concept.**

**The name of the algorithm comes from the**[**round-robin**](https://en.wikipedia.org/wiki/Round-robin_(disambiguation))**principle known from other fields, where each person takes an equal share of something in turn.**

**To schedule processes fairly, a round-robin scheduler generally employs**[**time-sharing**](https://en.wikipedia.org/wiki/Time-sharing)**, giving each job a time slot or *quantum***[**[4]**](https://en.wikipedia.org/wiki/Round-robin_scheduling#cite_note-McConnell2004-4)**(its allowance of CPU time), and interrupting the job if it is not completed by then. The job is resumed next time a time slot is assigned to that process. If the process terminates or changes its state to waiting during its attributed time quantum, the scheduler selects the first process in the ready queue to execute. In the absence of time-sharing, or if the quanta were large relative to the sizes of the jobs, a process that produced large jobs would be favored over other processes.**

**Round-robin algorithm is a pre-emptive algorithm as the scheduler forces the process out of the CPU once the time quota expires.**

**For example, if the time slot is 100 milliseconds, and *job1* takes a total time of 250 ms to complete, the round-robin scheduler will suspend the job after 100 ms and give other jobs their time on the CPU. Once the other jobs have had their equal share (100 ms each), *job1* will get another allocation of**[**CPU**](https://en.wikipedia.org/wiki/CPU)**time and the cycle will repeat. This process continues until the job finishes and needs no more time on the CPU.**

Question 2. Write the algorithm for proposed solution of the assigned problem.

Solution: **\* CPU scheduler picks the process from the circular/ready queue, set a timer to interrupt it after 1 time slice    / quantum and dispatches it .  
  
\* If process has burst time less than 1 time slice/quantum  
                  
             > Process will leave the CPU after the completion  
             > CPU will proceed with the next process in the ready queue / circular queue.  
  
    Else If process has burst time longer than 1 time slice/quantum  
  
             > Timer will be stopped. It cause interruption to the OS.  
             >   Executed process is then placed at the tail of the circular / ready queue by applying the context switch  
             > CPU scheduler then proceeds by selecting the next process in the ready queue.** **Here, User can calculate the average turnaround time and average waiting time along with the starting and finishing time of each process.**

**Turnaround time   :  It’s the total time taken by the process between starting and the completion  
  
waiting time         : It’s the time for which process is ready to run but not executed by CPU scheduler.**

Question 3. Calculate complexity of implemented algorithm.

Answer: **The time complexity of round-robin scheduling algorithms is O(1). It is easy to realize and is suitable to use in high-speed networks. One of the most classical is DRR scheduling algorithm. But DRR algorithm has some shortcomings that its delay characteristic is not very ideal and its output burst is big.**

Question 4. Explain all the constraints given in the problem.

Answer: **Constraint 1: Multiple queries are coming it can be of teacher or student-**

**CODE SNIPPET:**

while (1){

int detail;

printf("\nEnter your Occupation : \n1.Faculty \t2.Student \t3.Exit\n");

scanf("%d",&detail);

switch(detail){

case 1:

printf("\n\t\tFaculty Queries : ");

handleQueries();

break;

case 2:

printf("\n\t\tStudent Queries");

handleQueries();

break;

case 3:

printf("\nProgram terminated ");

exit(0);

break;

default:

printf("\nWrong key pressed");

break;

}

**Constraint 2: Sudesh Sharma can only contribute 1 hour for answering the queries.**

**CODE SNIPPET:**

int handleQueries(){

int i,j,n,time,remain,flag=0,timeQuant,detail;

int waitTime=0,turnaroundTime=0;

int at[10],bt[10],rt[10];

printf("\nEnter Total Process:\t ");

scanf("%d",&n);

remain=n;

for(i=0;i<n;i++)

{

printf("\nRequest Number %d :",i+1);

printf("\nEnter Arrival Time and Burst Time of Request");

scanf("%d",&at[i]);

scanf("%d",&bt[i]);

rt[i]=bt[i];

}

printf("\nHow much time each request take ? :\t");

scanf("%d",&timeQuant);

printf("\n\nRequest\t|Turnaround Time|Waiting Time\n\n");

for(time=0,i=0;remain!=0;)

{

if(rt[i]<=timeQuant && rt[i]>0)

{

time+=rt[i];

rt[i]=0;

flag=1;

}

else if(rt[i]>0)

{

rt[i]-=timeQuant;

time+=timeQuant;

}

if(rt[i]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",i+1,time-at[i],time-at[i]-bt[i]);

waitTime+=time-at[i]-bt[i];

turnaroundTime+=time-at[i];

flag=0;

}

if(i==n-1)

i=0;

else if(at[i+1]<=time)

i++;

else

i=0;

}

printf("\nAverage Waiting Time= %f\n",waitTime\*1.0/n);

printf("Avg Turnaround Time = %f",turnaroundTime\*1.0/n);

return 0;

}

Question 5: Explain the boundary conditions of the implemented code.

Answer: **Boundary conditions applied in the code is that the query handling time should not exceed 1 hr.**

**CODE SNIPPET:**

int handleQueries(){

int i,j,n,time,remain,flag=0,timeQuant,detail;

int waitTime=0,turnaroundTime=0;

int at[10],bt[10],rt[10];

printf("\nEnter Total Process:\t ");

scanf("%d",&n);

remain=n;

for(i=0;i<n;i++)

{

printf("\nRequest Number %d :",i+1);

printf("\nEnter Arrival Time and Burst Time of Request");

scanf("%d",&at[i]);

scanf("%d",&bt[i]);

rt[i]=bt[i];

}

printf("\nHow much time each request take ? :\t");

scanf("%d",&timeQuant);

printf("\n\nRequest\t|Turnaround Time|Waiting Time\n\n");

for(time=0,i=0;remain!=0;)

{

if(rt[i]<=timeQuant && rt[i]>0)

{

time+=rt[i];

rt[i]=0;

flag=1;

}

else if(rt[i]>0)

{

rt[i]-=timeQuant;

time+=timeQuant;

}

if(rt[i]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",i+1,time-at[i],time-at[i]-bt[i]);

waitTime+=time-at[i]-bt[i];

turnaroundTime+=time-at[i];

flag=0;

}

if(i==n-1)

i=0;

else if(at[i+1]<=time)

i++;

else

i=0;

}

printf("\nAverage Waiting Time= %f\n",waitTime\*1.0/n);

printf("Avg Turnaround Time = %f",turnaroundTime\*1.0/n);

return 0;

}

Question 6. Explain all the test cases applied on the solution of assigned problem.

Answer: **TEST CASE 1-**

**PRE CONDITION:**

**User has to select an option from the given to choose his/her occupation-**

**1>Faculty 2.>Student 3>Exit**

|  |  |  |  |
| --- | --- | --- | --- |
| **S NO.** | **INPUT** | **EXPECTED OUTPUT** | **VALID/INVALID** |
| **1.** | **1** | **OCCUPATION: FACULTY** | **VALID** |
| **2.** | **2** | **OCCUPATION: STUDENT** | **VALID** |
| **3** | **3** | **YOU ARE EXITING THE SYSTEM** | **VALID** |
| **4** | **ANY OTHER KEY** | **WRONG KEY PRESSED** | **INVALID** |

**TEST CASE 2-**

**PRE CONDITION:**

**The Linux expert can’t handle the query for more than 1 hr.**

|  |  |  |  |
| --- | --- | --- | --- |
| **S NO.** | **INPUT** | **EXPECTED OUTPUT** | **VALID/INVALID** |
| **1.** | **Time <1hr** | **QUERY HANDLING CONTINUES** | **VALID** |
| **2.** | **Time>1hr** | **YOUR TIME IS OVER** | **INVALID** |